



Available Online at www.hithaldia.in/locate/ECCN
All Rights Reserved

ORIGINAL CONTRIBUTION

ScriptGenie AI: A Groq API and LLaMA 3-Based Full-Stack Framework for Automated YouTube Script Generation

¹Kumar Harsh, ²MD Tayyeb Hussain, ³Satyam Kumar, ⁴Jayanta Kumar Bag, ⁵Tilak Mukherjee

¹²³⁴*Dept. of Electronics & Communication Engineering, Haldia Institute of Technology, Haldia, Purba Medinipur, West Bengal, India*

¹*UG Student, Department of ECE, Haldia Institute of Technology, Haldia, West Bengal*

²*Assistant Professor, Department of ECE, Haldia Institute of Technology, West Bengal, India*

ABSTRACT

Scripting remains one of the most time-consuming stages involving YouTube video production, requiring creators to structure a hook, introduction, main content, and call-to-action while also drafting titles, tags, and descriptions for discoverability. ScriptGenie AI is a full-stack web application that automates this process by converting a single user-supplied topic into a complete, structured video script. The system integrates a vanilla HTML/CSS/JavaScript front end with a Node.js and Express.js back end, which communicates with the Groq inference API so as to query the open-weight LLaMA 3 language model through a structured, JSON-constrained prompt. The present paper details the system architecture, the rationale for selecting an inference-API-based design over local model hosting, the request-response workflow, and the prompt-engineering strategy used to obtain reliably structured output.

KEYWORDS: YouTube Script Generation, Large Language Models, Groq API, LLaMA 3, Prompt Engineering; MERN-based Web Architecture, MongoDB Atlas.

1. INTRODUCTION

The growth of YouTube as a primary medium for information and entertainment has made consistent content output a central challenge for all class of creators. Producing a video typically begins with scripting: framing an attention-grabbing hook, organizing the main content into coherent sections, and closing with a call-to-action, in addition to preparing supporting metadata such as titles, tags, and descriptions for search visibility. Manually performing these sub-tasks for every video is repetitive and time-intensive task, particularly for independent creators managing the entire production pipeline alone.

However, recent advances in large language models (LLMs) have made it feasible to automate substantial portions of this writing workload through natural-language prompting rather than task-specific model training. Cloud-hosted inference APIs now allow small teams to access such models without owning the underlying infrastructure, shifting the

engineering effort from model development to system integration, prompt design, and workflow orchestration.

This paper presents ScriptGenie AI, a full-stack tool that accepts a topic, a desired tone, a target length, and an audience type, and returns a structured script object comprising a hook, introduction, main content sections, a call-to-action, title suggestions, tags, and a description.

The system does not train or fine-tune any language model; instead, it is built entirely around the Groq inference API serving the openly released LLaMA 3 model. The contribution of this work is therefore an applied system design and integration study. It documents a kind of architecture, an API-selection rationale, and a prompt-engineering approach for structured content generation, rather than a new model or a benchmarked algorithm.

2. RELATED WORK

The use of generative AI in video content creation has catapulted growing research attention and growth. Analysis of a large corpus of YouTube how-to videos and related search reported that creators already use general-purpose LLMs at multiple stages of production, including topic ideation, script drafting, and title generation [1, 2]. Broader surveys of content creators similarly report that generative tools are being adopted for writing and editing tasks, while also raising concerns around originality and disclosure that motivate careful, transparent tool design [2]. Distinct from tools that summarize or repurpose existing long-form media, ScriptGenie AI addresses the earlier stage of the pipeline: producing an original script structure from a bare topic before any recording takes place [1,2].

It needs to be mentioned here that the system relies entirely on prompting a pretrained model rather than on fine-tuning, and its design is informed by the broader literature on prompt engineering. A wide range of prompting techniques for eliciting structured, task-specific behaviour from LLMs without parameter updates have been reported [3], which motivates the JSON-schema-constrained prompting strategy adopted in this work. On the inference side, the system depends on Groq's Language Processing Unit (LPU), a deterministic, statically scheduled accelerator architecture purpose-built for low-latency LLM inference, as distinct from general-purpose GPU pipelines [4, 5]. Finally, the underlying language model, LLaMA 3, is integral part of an openly released model family whose architecture and training methodology are documented by Meta AI [6]. The backend integration pattern used here, an Express.js REST layer over a MongoDB document store, follows established practices for JSON-native web-service architecture systems [7].

3. SYSTEM ARCHITECTURE

ScriptGenie AI follows a three-tier architecture consisting of a presentation layer, layer, and

persistence layer application, which are connected through a REST interface.

3.1 Presentation Layer

The client is implemented in plain HTML5, CSS3, and vanilla JavaScript, avoiding a front-end framework dependency. It captures the user's topic, tone (professional, funny, storytelling, motivational, educational, or casual), desired length (shorts, standard, or long form), and target audience; and sends this payload to the backend; and thereby renders the returned script into labelled sections (hook, intro, main content, call-to-action), a metadata tab (tags and description), and a titles tab. The interface also exposes copy-to-clipboard and .txt download actions.

3.2 Application Layer

The backend is built on Node.js with the Express.js framework and is organized using a routes-controllers-services pattern. The route `scriptRoutes.js` exposes `POST /api/scripts/generate`, which is handled by `scriptController.js`. This controller essentially validates the incoming topic length and confirms that the Groq API key is configured before delegating generation to a dedicated service module, `openaiService.js`, which constructs the model prompt and issues the request to the Groq API. A parallel route, `savedRoutes.js`, exposes CRUD endpoints for the script library, handled by `savedController.js`.

3.3 Persistence Layer

Saved scripts are stored in MongoDB Atlas, a cloud-hosted, fully managed instance of MongoDB, and accessed through the Mongoose object-modeling library. A `Script` schema (`Script.js`) defines the structure of a persisted script document, enabling later retrieval and deletion through the `/api/saved` endpoints without requiring a locally administered database server.

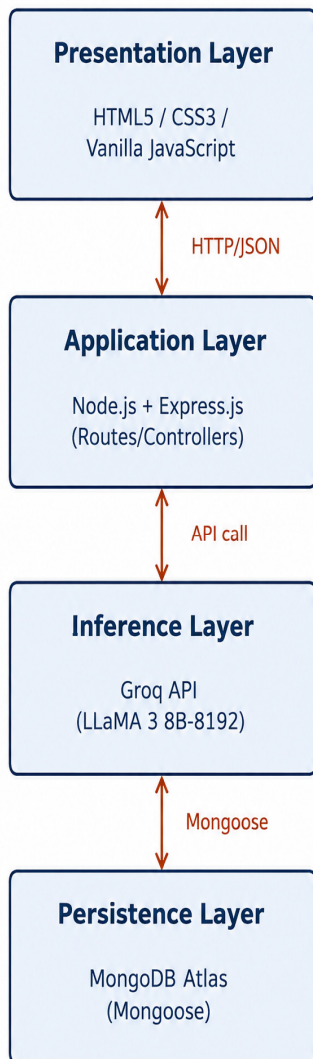


Figure 1: Three-tier system architecture of ScriptGenie AI, showing the presentation, application, inference, and persistence layers and their REST/API communication paths

4. METHODOLOGY AND API INTEGRATION

4.1 Rationale for API-Based Inference

ScriptGenie AI deliberately does not host or fine-tune a language model. Locally hosting an LLM of comparable capability would require dedicated GPU infrastructure and ongoing maintenance, which is impractical for a small student-built application. The Groq API was selected instead of a locally hosted alternative

for three reasons drawn directly from the project's requirements:

- i. It offers a free tier sufficient for demonstration and iterative development.
- ii. It serves the openly released LLaMA 3 (llama3-8b-8192) model, avoiding dependence on a closed-source provider.
- iii. It is built on Groq's LPU accelerator, whose statically schedule and deterministic pipeline architecture is reported to substantially reduce per-token inference latency relative to general-purpose GPU serving [4, 5], which is important for an interactive, request-response user experience.

4.2 Request-Response Flow

A generation request follows a linear pipeline: the client validates and submits the topic, tone, length, and audience fields; the Express route forwards the request to scriptController.js, which re-validates the payload server-side; openaiService.js assembles a structured prompt embedding these fields and instructs the model to respond strictly in a predefined JSON schema (hook, intro, mainContent, cta, title, tags, description); this prompt is sent to the Groq /chat/completions-equivalent endpoint targeting llama3-8b-8192 with a temperature of 0.8 and a maximum token budget of 2000; the returned JSON is parsed and returned to the client, which renders it into the appropriate UI sections.

Here the user chooses to save a script, a second, independent request persists the same JSON structure into the Script collection in MongoDB Atlas.

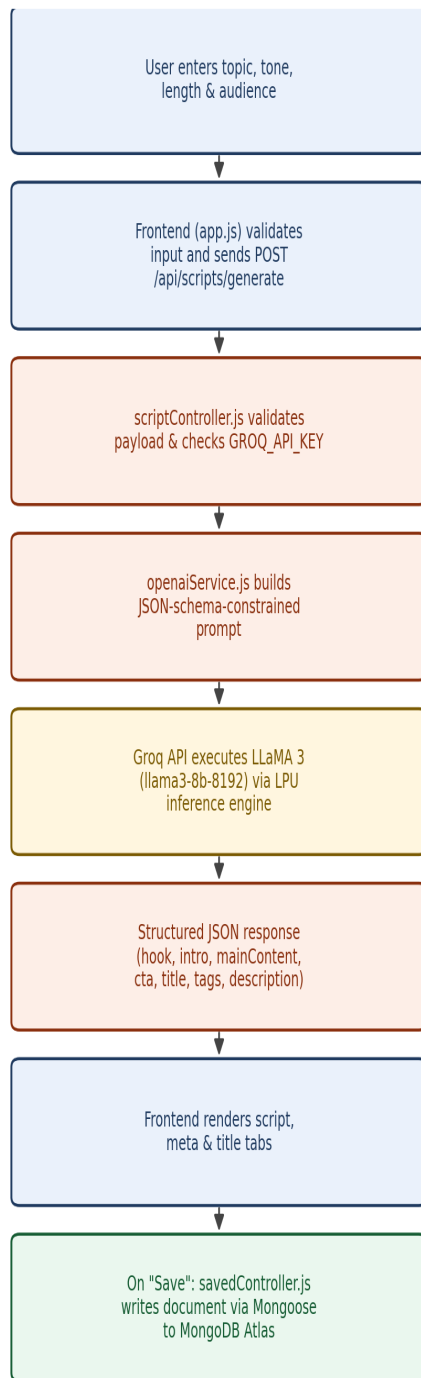


Figure 2: Request-response workflow of a script-generation call, from user input through Groq/LLaMA 3 inference to optional persistence in MongoDB Atlas.

4.3 Prompt Design

Because the system has no control over model weights, output reliability depends entirely on prompt construction. The prompt explicitly role-primers the model as a professional YouTube scriptwriter, supplies the topic, tone, a length-specific structural guide, and audience descriptor, and constrains the response format to a fixed JSON object.

This schema-constrained prompting approach follows established practice for eliciting structured, pursuable output from general-purpose LLMs without any fine-tuning step [3], and allows the front end to deterministically map model output to fixed UI fields.

5. RESULTS AND DISCUSSIONS

As ScriptGenie AI does not train a model or operate on a labelled dataset, no accuracy, precision, or benchmark metrics are applicable or reported. The system was instead evaluated functionally: the generation endpoint was exercised across the six supported tones and three supported length options, and in each case returned a syntactically valid JSON object containing all expected fields, which the front end successfully rendered without manual reformatting. The save, retrieve, and delete operations against the MongoDB Atlas-backed library were similarly verified to behave as CRUD operations should, that is, saved entries reappear correctly on retrieval and are removed on deletion.

Consistent with Groq's documented LPU-based serving characteristics [4, 5], generation requests were observed to return within a short, interactive time window, in line with the project's own documented expectation of roughly one to two seconds end-to-end. This should be read as a qualitative operational characteristic of the deployed system rather than a controlled, statistically validated latency benchmark, since no repeated-trial timing study was conducted. Likewise, the Groq free tier's daily request allowance was found sufficient for iterative development and demonstration use but was not stress-tested under sustained concurrent load, which is noted as a major limitation below.

6. LIMITATIONS AND FUTURE SCOPE

The system's output quality is bounded by the capability of the llama3-8b-8192 model and by prompt design rather than by any mechanism the system itself learns or adapts. Therefore, it inherits general LLM limitations such as occasional factual inaccuracy in generated content, which is not independently fact-checked before being returned to the user. Reliance on a third-party inference API also introduces an external dependency in terms of service availability, rate limits, and pricing policy changes on Groq's side that directly affect system availability. Finally, the current implementation has not been evaluated under multi-user concurrent load or adversarial input conditions.

Several extensions have been planned as future scope of work. Voice-over generation would convert a produced script directly into narrated audio, closing the gap between scripting and recording. Multi-language support would allow script generation in languages beyond English, broadening the tool's creator base. An AI-assisted thumbnail generator would extend the pipeline into the visual domain. A video-SEO optimizer could automate title and tag refinement using performance feedback, while an analytics dashboard could track how generated scripts correlate with published video performance. Direct export integrations with social platforms would additionally streamline distribution of generated titles and descriptions beyond YouTube.

7. CONCLUSION

This paper presented ScriptGenie AI, a full-stack web application that automates YouTube script generation by integrating a vanilla JavaScript front end, an Express.js/Node.js backend, the Groq inference API serving LLaMA 3, and a MongoDB Atlas persistence layer. Rather than proposing a new model or algorithm, the contribution of this work particularly lies in its system architecture, its rationale for choosing an API-based inference strategy over local model

hosting, and its schema-constrained prompt design for reliable structured generation.

Functional testing confirmed that the generation, storage, and retrieval pipeline operates as designed, though the evaluation remains qualitative in the absence of a training dataset or benchmarked task. Furthermore, future work shall extend the system toward multilingual, multimedia, and analytics-driven capabilities as outlined above.

REFERENCES

- [1] Niu, S., et al. (2025). Making AI-Enhanced Videos: Analyzing Generative AI Use Cases in YouTube Content Creation. Extended Abstracts of the CHI Conference on Human Factors in Computing Systems. arXiv:2503.03134.
- [2] Content Creation with Generative AI: How Do Content Creators Responsibly Use Generative AI Tools? (2025). Proceedings of the ACM on Human-Computer Interaction. doi:10.1145/3788080.
- [3] Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. arXiv:2402.07927.
- [4] Groq Inc. (2024). What is a Language Processing Unit? GroqCloud Technical Documentation. Available: <https://groq.com/blog/the-groq-lpu-explained>.
- [5] AI Accelerators for Large Language Model Inference: Architecture Analysis and Scaling Strategies. (2025). arXiv:2506.00008.
- [6] Meta AI, Llama Team. (2024). The Llama 3 Herd of Models. arXiv:2407.21783.
- [7] Holmes, & Harber, A. Getting MEAN with Mongo, Express, Angular, and Node.js, 2nd Edition. Manning Publications. (2019).